(FILE 'USPAT' ENTERED AT 08:57:59 ON 16 APR 1999)

L1	5	S	AUTHENTICAT? (P) APPLET#
L2	179	S	APPLET#
L3	34	S	AUTHENTICAT? AND L2
L4	13	S	GLOBAL SERVE?
L5	0	S	L1 AND L4
L6	0	S	L2 AND L4

(FILE 'USPAT' ENTERED AT 12:52:52 ON 15 APR 1999)
L1 21 S NEGOTIAT? (P) PROTOCOL# (P) ENCRYPT?
L2 1 S NEGOTIAT? (P) (PROTOCOL# (2A) ENCRYPT?)

=> d l1 2,5,11,12,16-19

2. 5,825,890, Oct. 20, 1998, Secure socket layer application program apparatus and method; Taher Elgamal, et al., 380/49; 709/302 [IMAGE AVAILABLE]

5. 5,796,836, Aug. 18, 1998, Scalable key agile cryptography; Thomas R. Markham, 380/28, 37, 42, 43, 46 [IMAGE AVAILABLE]

11. 5,657,390, Aug. 12, 1997, Secure socket layer application program apparatus and method; Taher Elgamal, et al., 380/49, 4, 25 [IMAGE AVAILABLE]

12. 5,638,448, Jun. 10, 1997, Network with secure communications sessions; Minhtam C. Nguyen, 380/29, 9, 21, 23, 25, 37, 43, 49 [IMAGE AVAILABLE]

16. 5,412,730, May 2, 1995, Encrypted data transmission system employing means for randomly altering the encryption keys; Michael F. Jones, 380/46, 9, 21, 29, 49 [IMAGE AVAILABLE]

17. 5,371,794, Dec. 6, 1994, Method and apparatus for privacy and authentication in wireless networks; Whitfield Diffie, et al., 380/21, 30 [IMAGE AVAILABLE]

18. 5,029,206, Jul. 2, 1991, Uniform interface for cryptographic services; Joseph T. Marino, Jr., et al., 380/4, 45 [IMAGE AVAILABLE]

19. 5,010,572, Apr. 23, 1991, Distributed information system having automatic invocation of key management negotiations protocol and method; Erwin W. Bathrick, et al., 380/21, 23 [IMAGE AVAILABLE]

=> d 2 hit

US PAT NO: 5,825,890 [IMAGE AVAILABLE] L1: 2 of 21

SUMMARY:

BSUM(10)

In another aspect, the invention provides a more efficient handshake **protocol** and session key generation scheme. When a client and server application first establish a secure sockets connection, in accordance with the invention, they engage in a novel handshake **protocol** in which they **negotiate** security procedures, produce a master key and generate session keys to be used to **encrypt** and decrypt information transferred through the sockets connection. If there are multiple connections between the client and server applications during a prescribed time interval, then the handshake **protocol** may elect to reuse a previously **negotiated** master key, thereby obviating the need to generate a new master key, and saving time in establishing a secure connection.

DETDESC:

DETD(26)

Referring to FIG. 4, there is shown the message flow during handshake **protocol negotiation** where RSA key exchange is employed and no "session-identifier" is stored in server cache. Note that at the stage at which the message exchange in FIG. 4 begins, the client and server already have established a "sockets" connection between them, and the server has determined that the connection is to be a secure connection that employs the novel SSL processes and program control mechanisms described herein. As explained above, the sockets API is well known. Moreover, in accordance with the invention, the sockets connection is initiated by a client application SSL.sub.-- open call to its SSL library which is bound to the client application. The data transferred between client and server can be **encrypted**/decrypted as it is channeled through the socket. But first, before any data is transferred, the client and the server must **negotiate** an **encryption** technique for the data transfer, authenticate the connected parties (server and possibly client too), and check the integrity of the agreed upon secure connection. This **negotiation** is carried out in response to the SSL.sub.-- open call through message flow of FIG. 4, which takes place using the sockets connection previously set up as part of that same SSL.sub.-- open call.

DETDESC:

DETD(30)

Once the master key has been delivered to the server, the server and the client both can independently generate the session keys used to actually **encrypt**/decrypt data transferred following successful completion of the handshake **protocol**. The session keys are produced using well known techniques such as through hash functions referenced in the sections below or some other function of the master key and another data value. A more detailed explanation of the session key production

techniques used in the presently preferred embodiment of the invention is provided below in the handshake **protocol** section. It should be understood that, while public key **encryption** techniques are used for master key exchange, the actual **encryption**/decryption of data transferred between client and server through the socket is achieved using a well known bulk cipher, such as RC2, RC4 or IDEA, **negotiated** through the respective cipher specs components of the information exchanged in the client-hello and server-hello messages. The selected bulk cipher uses the session keys to encipher/decipher data and messages transferred through the socket connection.

DETDESC:

DETD(31)

The client sends a client-finished message which indicates that the client is satisfied with the server. In a present embodiment of the invention, the client-finish message includes a hash of all of the handshake **protocol** messages previously sent by the client. The hash is **encrypted** using the agreed upon bulk cipher plus a session key generated by the client, referred to in FIG. 4 as the client-write-key. Note that the hash function also is **negotiated** as part of the cipher.sub.-- specs. In addition, the connection-identification previously sent to the client by the server is transmitted to the server with the client-finish message to authenticate the channel. The server uses the client messages handshake hash to verify the integrity of the communication between client and server. This final integrity check will expose third party intervention even if it occurred at the beginning of the handshake **protocol**.

DETDESC:

DETD(32)

The server sends a server-finished message which indicates that the server is satisfied with the client and is ready to begin the actual data transfer through the socket connection. In a present embodiment of the invention, the server-finish message includes a hash of all of the handshake **protocol** messages previously sent by the server. The hash is **encrypted** using the agreed upon bulk cipher plus a session key generated by the server, referred to in FIG. 4 as the server.sub.-- write.sub.-- key. The hash function also is **negotiated** as part of the cipher.sub.-- specs. The server uses the server messages handshake hash to verify the integrity of the communication between client and server as explained above.

DETDESC:

DETD(40)

For instance, an electronic document containing hyperlinks may be displayed by the client browser application. Whenever a user "clicks" on (or selects) portion of the document associated with a hyperlink, another electronic document or file or a graphic or some other remotely stored information that corresponds to the link is retrieved over the Internet from the server. Before the transfer can occur, however, the handshake **protocol** is **negotiated**, and **encryption**/decryption information is developed as described above. That information is stored with the session.sub.-- identification for that connection. When that new document (or other information) has been transferred to the client, the SSL library causes the secure sockets connection used to accomplish the transfer of that next document to be closed. The session.sub.-- identification information (master key, block cipher, hash function) is stored in cache by the client and server applications for a prescribed time intervals. If within that time a user of that same client browser

SUMMARY:

BSUM(10)

In another aspect, the invention provides a more efficient handshake **protocol** and session key generation scheme. When a client and server application first establish a secure sockets connection, in accordance with the invention, they engage in a novel handshake **protocol** in which they **negotiate** security procedures, produce a master key and generate session keys to be used to **encrypt** and decrypt information transferred through the sockets connection. If there are multiple connections between the client and server applications during a prescribed time interval, then the handshake protcol may elect to re-use a previously **negotiated** master key, thereby obviating the need to generate a new master key, and saving time in establishing a secure connection.

DETDESC:

DETD(26)

Referring to FIG. 4, there is shown the message flow during handshake **protocol negotiation** where RSA key exchange is employed and no "session-identifier" is stored in server cache. Note that at the stage at which the message exchange in FIG. 4 begins, the client and server already have established a "sockets" connection between them, and the server has determined that the connection is to be a secure connection that employs the novel SSL processes and program control mechanisms described herein. As explained above, the sockets API is well known. Moreover, in accordance with the invention, the sockets connection is initiated by a client application SSL.sub.-- open call to its SSL library which is bound to the client application. The data transferred between client and server can be **encrypted**/decrypted as it is channeled through the socket. But first, before any data is transferred, the client and the server must **negotiate** an **encryption** technique for the data transfer, authenticate the connected parties (server and possibly client too), and check the integrity of the agreed upon secure connection. This **negotiation** is carried out in response to the SSL.sub.-- open call through message flow of FIG. 4, which takes place using the sockets connection previously set up as part of that same SSL.sub.-- open call.

DETDESC:

DETD(30)

Once the master key has been delivered to the server, the server and the client both can independently generate the session keys used to actually **encrypt**/decrypt data transferred following successful completion of the handshake **protocol**. The session keys are produced using well known techniques such as through hash functions referenced in the sections below or some other function of the master key and another data value. A more detailed explanation of the session key production techniques used in the presently preferred embodiment of the invention is provided below in the handshake **protocol** section. It should be understood that, while public key **encryption** techniques are used for master key exchange, the actual **encryption**/decryption of data transferred between client and server through the socket is achieved using a well known bulk cipher, such as RC2, RC4 or IDEA, **negotiated** through the respective cipher.sub.-- specs components of the information exchanged in the client-hello and server-hello messages. The selected bulk cipher uses the session keys to encipher/decipher data and messages transferred through the socket connection.

application "clicks" on another hyperlink then the client browser application will set up another sockets connection with the server to satisfy this latest request. Rather than go through the entire time consuming handshake **protocol**, however, the client and server will use the previously generated and stored session.sub.-- identification information (master key, block cipher, hash function) to secure data transfer through the new socket set up by the SSL library to satisfy this latest request.

DETDESC:

DETD(78)

When the client and server operating system services have established the socket connection described above, the client operating system returns a message to the SSL library indicating successful connection. The client side SSL library and the server side SSL library then engage in a handshake **protocol** in accordance with the invention using the socket connection that has been established by the client and server operating system services. During the handshake procedure, for example, **encryption**/decryption precautions are **negotiated**, the server is authenticated and the integrity of the socket security is checked. A detailed explanation of the handshake **protocol** is provided elsewhere in this specification. Once the handshake procedure has been successfully completed, the client SSL library returns to the client application a message indicating the HTTP server endpoint of the socket connection. Meanwhile, the server side SSL library listens on the endpoint port.

=> d 5 hit

US PAT NO: 5,796,836 [IMAGE AVAILABLE] L1: 5 of 21

DETDESC:

DETD(40)

In the system of FIG. 9, devices 106 and 110 exchange cryptographic identities and **negotiate** security services when establishing the ATM connection. They then provide ATM speed **encryption** and decryption for the devices they connect to ATM network 108. The only **protocol** overhead after the secure connection has been established is synchronization information and the input devices such as devices 102 and 112 need not be aware that cryptographic services are being provided. (In one embodiment, cells arriving on virtual circuits for which no secure ATM connection has been established will be dropped.)

DETDESC:

DETD(42)

In one embodiment, device 106 maintains information about a connection setup request for use in subsequent connection setup or release processing. Subsequent to ATM connection setup with a peer device (such as device 110), **protocol** exchanges occur between devices 106 to exchange cryptographic keys and **negotiate** security services to setup a secure ATM connection. Once the ATM connection is made, however, ATM cells flowing through device 106 come up only as far as ATM layer 120 before being **encrypted** or decrypted and sent out the appropriate interface.

=> d 11 hit

US PAT NO: 5,657,390 [IMAGE AVAILABLE] L1: 11 of 21

DETDESC:

DETD(31)

The client sends a client-finished message which indicates that the client is satisfied with the server. In a present embodiment of the invention, the client-finish message includes a hash of all of the handshake **protocol** messages previously sent by the client. The hash is **encrypted** using the agreed upon bulk cipher plus a session key generated by the client, referred to in FIG. 4 as the client-write-key. Note that the hash function also is **negotiated** as part of the cipher.sub.-- specs. In addition, the connection-identification previously sent to the client by the server is transmitted to the server with the client-finish message to authenticate the channel. The server uses the client messages handshake hash to verify the integrity of the communication between client and server. This final integrity check will expose third party intervention even if it occurred at the beginning of the handshake **protocol**.

DETDESC:

DETD(32)

The server sends a server-finished message which indicates that the server is satisfied with the client and is ready to begin the actual data transfer through the socket connection. In a present embodiment of the invention, the server-finish message includes a hash of all of the handshake **protocol** messages previously sent by the server. The hash is **encrypted** using the agreed upon bulk cipher plus a session key generated by the server, referred to in FIG. 4 as the server.sub.-- write.sub.-- key. The hash function also is **negotiated** as part of the cipher.sub.-- specs. The server uses the server messages handshake hash to verify the integrity of the communication between client and server as explained above.

DETDESC:

DETD(40)

For instance, an electronic document containing hyperlinks may be displayed by the client browser application. Whenever a user "clicks" on (or selects) portion of the document associated with a hyperlink, another electronic document or file or a graphic or some other remotely stored information that corresponds to the link is retrieved over the Internet from the server. Before the transfer can occur, however, the handshake **protocol** is **negotiated**, and **encryption**/decryption information is developed as described above. That information is stored with the session.sub.-- identification for that connection. When that new document (or other information) has been transferred to the client, the SSL library causes the secure sockets connection used to accomplish the transfer of that next document to be closed. The session.sub.-- identification information (master key, block cipher, hash function) is stored in cache by the client and server applications for a prescribed time intervals. If within that time a user of that same client browser application "clicks" on another hyperlink, then the client browser application will set up another sockets connection with the server to satisfy this latest request. Rather than go through the entire time consuming handshake **protocol**, however, the client and server will use the previously generated and stored session.sub.-- identification information (master key, block cipher, hash function) to secure data transfer through the new socket set up by the SSL library to satisfy this latest request.

DETDESC:

When the client and server operating system services have established the socket connection described above, the client operating system returns a message to the SSL library indicating successful connection. The client side SSL library and the server side SSL library then engage in a handshake **protocol** in accordance with the invention using the socket connection that has been established by the client and server operating system services. During the handshake procedure, for example, **encryption**/decryption precautions are **negotiated**, the server is authenticated and the integrity of the socket security is checked. A detailed explanation of the handshake **protocol** is provided elsewhere in this specification. Once the handshake procedure has been successfully completed, the client SSL library returns to the client application a message indicating the HTTP server endpoint of the socket connection. Meanwhile, the server side SSL library listens on the endpoint port.

US PAT NO: 5,638,448 [IMAGE AVAILABLE] L1: 12 of 21

DETDESC:

DETD(140)

After decrypting the third logon packet, the client now saves the session key Ks and the IV in its own memory for future communication with the server. To improve performance, the client and server build **encryption** and decryption key schedules of s.sup.3 DES and stores them in the session information block structure as shown in FIGS. 5A-B. The S-boxes of s.sup.3 DES are also combined with the P-box of DES algorithm to further enhance **encryption** speed. As described so far, the authentication techniques not only provide a secure identification procedure, but also a secure **negotiation protocol** to set up communication sessions (i.e, **encryption** method, compression method, operating platform, language, etc.).

> d l1 16 hit

US PAT NO: 5,412,730 [IMAGE AVAILABLE] L1: 16 of 21

DETDESC:

DETD(48)

To make a secured transmission, the calling station uses PHONENUM to establish the connection, normal modem handshaking procedures are executed to establish a working data connection, including standard parameter **negotiations** (e.g. the V.42 parameters if that **protocol** is being used). If the security key is enabled, and a secure transmission is being requested by the caller, the answering modem will not send its parameter message (the V.42 XID frame) until it receives one from the originator, this initial message including the (unencrypted) originator's serial number. The answering modem uses the received serial number to select the locally stored **encryption** key corresponding to that serial number, and **encrypts** its responsive XID frame using the fetched key. Thereafter, all transmissions between the originating and answering modems are **encrypted** and the **encryption** keys at each end of the secure ling are thereafter altered in accordance with the **encryption** algorithm as heretofore described.

=> d l1 17 hit

US PAT NO: 5,371,794 [IMAGE AVAILABLE] L1: 17 of 21

DETDESC:

DETD(23)

Having obtained a certificate for each machine, as well as secure backup of the private keys, the mobile and base are in a position to engage in a secure **protocol**. The two parties exchange certificates and engage in a mutual challenge-response **protocol**. The **protocol** allows **negotiation** of the shared key algorithm. This allows for enhancing the **protocol** in the future to better shared key cryptosystems and also allows for interoperability between US and non-US versions of the product, should they require different **encryption** algorithms for exportability purposes.

=> d l1 18 hit

US PAT NO: 5,029,206 [IMAGE AVAILABLE] L1: 18 of 21

DETDESC:

DETD(22)

The KMP of the KMASE 30 provides for option **negotiation** to allow for determination of the security attributes associated with a particular traffic **encryption** key. The security attributes that may be **negotiated** include the security services that a particular traffic **encryption** key may provide, the security **protocol** that will use the particular traffic **encryption** key, the possible security labels assigned to protected data, and the granularity of the particular traffic **encryption** key assignment. The particular traffic **encryption** key is used to **encrypt** the options being sent to the requesting network **encryption** device. The requesting network **encryption** device must correctly decrypt and verify the requested options before sending back an **encrypted** response. The KMP of KMASE 30 then decrypts and verifies the response from the remote network **encryption** device. The traffic **encryption** key is not valid for protection of user data traffic until the successful **negotiation** of all security attributes associated with that particular key.

=> d l1 19 hit

US PAT NO: 5,010,572 [IMAGE AVAILABLE] L1: 19 of 21

DETDESC:

DETD(9)

If the AIKM processor 14 does find a match, then a security key **protocol** request signal is generated to activate the key **negotiation** processor 20 and, in addition, suspends processing data units with the same address as the data unit in process. This will cause the data unit receive 15 to ignore further data units with the same address as the data unit in process until processing and transmission of the data unit. The key **negotiation** processor 20 now communicates with the key **negotiation** processor 24 of end system B via transmitter/receivers 18, 22 and communications medium 8. The two key **negotiations** processors 20, 24 perform a fully **encrypted** **negotiation** exchange to securely establish a security **protocol** key and **protocol** specification for the end-system B appropriate for the exchange of the data unit in process.

DETDESC:

DETD(13)

In response to the data transfer request signal, the next step in the method is to compare the data unit address with the end-system addresses, security keys and **protocol** specifications. In response to a match, a transmit enable signal is generated. In response to the absence of a match, the data unit address is compared with the stored PAS address. In

response to a match between addresses, an automatic invocation of key management request signal is generated. In response to this request to this request signal, an **encrypted** security key **negotiation** is performed and an appropriate security key is generated. Upon generation of the security key, a transfer enable signal is generated and a security **protocol** data transfer is performed. The security **protocol** processor 28 of end-system B receives the data unit from end-system A and applies the appropriate security **protocol** when decoding data. The data unit is then passed to the end-system B user.

CLAIMS:

CLMS(1)

What is claimed is:

1. In a distributed information system which includes a plurality of end-systems each of which includes data unit transmitting and receiving means having a security **protocol** and a corresponding security **protocol** key from one end-system to another end-system, the improvement comprising:

secure address storage means for storing a set of end-system addresses and corresponding end-system security **protocol** keys;

protocol address storage means for storing a set of end-system addresses requiring security **protocol** data transfers;

data unit receiver means for receiving a data unit and for generating a data unit transfer request signal which includes an end-system address and data unit security **protocol**;

intermediate storage means for storing a received data unit in response of a data transfer request signal and outputting the stored data unit to the end-system transmitting and receiving means in response to a transfer enable signal;

automatic key management processor means responsive to the data unit transfer request signal for comparing the received data unit security key and end-system address to the set of end system address and security **protocol** keys, and for generating the transfer enable signal in response to a match therebetween, and in the absence of a match therebetween, comparing the received data unit address to the end-system addresses from the **protocol** address storage means and generating a security key **protocol** request signal in response to a match therebetween, and for generating the transfer enable signal in response to the absence of a match between the data unit address and an address in the set of addresses stored in the **protocol** address storage means;

security key **negotiating** means responsive to the security key **protocol** request signal for **negotiating** a security key with another end-system and for generating a security key **negotiation** confirm signal upon completion of a **negotiation**;

means responsive to the security key **negotiation** confirm signal for storing the security key for the another end-system in the one system secure address store storage means;

and wherein each end-system is responsive to the security key **negotiation** confirm signal and is adapted to generate the transfer enable signal, and wherein the end-system data unit transmitting and receiving means are responsive to the transfer enable signal and are adapted to transfer the data unit corresponding to the security **protocol** of the received data unit, and wherein the security key **negotiation** comprises a fully **encrypted negotiation** exchange.

CLAIMS:

CLMS(4)

4. A method of automatic invoking secure communications between

end-systems of a distributed information system, said method comprising the steps of;

(a) storing a set of end-system addresses and corresponding security keys and security **protocol** specifications;

(b) storing a set of end-system addresses that includes the addresses of all end-systems requiring security **protocols** for a secure data transfer;

(c) generating a data that includes an end-system address and security **protocol** specification, and generating a data transfer request signal in response thereto;

(d) comparing the generated end-system address and **protocol** specification to the set of end-system addresses and **protocol** specifications in response to the data transfer signal, and generating a transmit enable signal in response to a match therebetween;

(e) comparing the data unit specification to the set of end-system addresses requiring a security **protocol**;

(f) generating an automatic invocation of key management request signal in response to the absence of a match between the data unit specification to the set of end-system addresses requiring a security **protocol**;

(g) performing an **encrypted** security key **negotiation** between the end-systems in response to the invocation of the key management request signal and generating a security key confirm signal in response to completion thereof;

(h) generating a transfer enable signal a response to the security key confirm signal;

(i) performing a security **protocol** data transfer in accordance with the appropriate security **protocol** specification in response to the transfer enable signal;

(j) generating the transfer enable signal in the absence of a match between the data unit address and an end-system address for which a security **protocol** is required;

(k) storing a **negotiated** security key in the set of stored end-system addresses and corresponding security keys and **protocol** specifications in response to the security key **negotiation** confirm signal; and

(l) storing a received data unit in an intermediate storage means during the performance of steps (b) through (g).